witnessCalc

A Tool for Calculating Responses of Witness Foils

By Robert Crabbs
8-10-2009

INTRODUCTION

In the interest of assaying spent nuclear fuel, we evaluate nuclear resonance florescence (NRF) as a possible means of identifying nuclides in a given sample. Every nuclide has a unique set of nuclear excitation energies, which can serve as a signature for the presence of that nuclide. In NRF interrogation, a continuous-spectrum photon beam is directed on a target. The photons that are very close to the excitation energies of the nuclides in the target are preferentially absorbed, and re-emitted a short time later as isotropic radiation of the same energy. Direct measurement of the output radiation at a heavily-shielded backwards angle isolates the NRF photons from the interrogating beam. The isolated NRF spectrum gives quantitative data about the composition of the target. However, NRF photons generated within the target will be heavily attenuated on the way out, resulting in a low count rates at the detector.

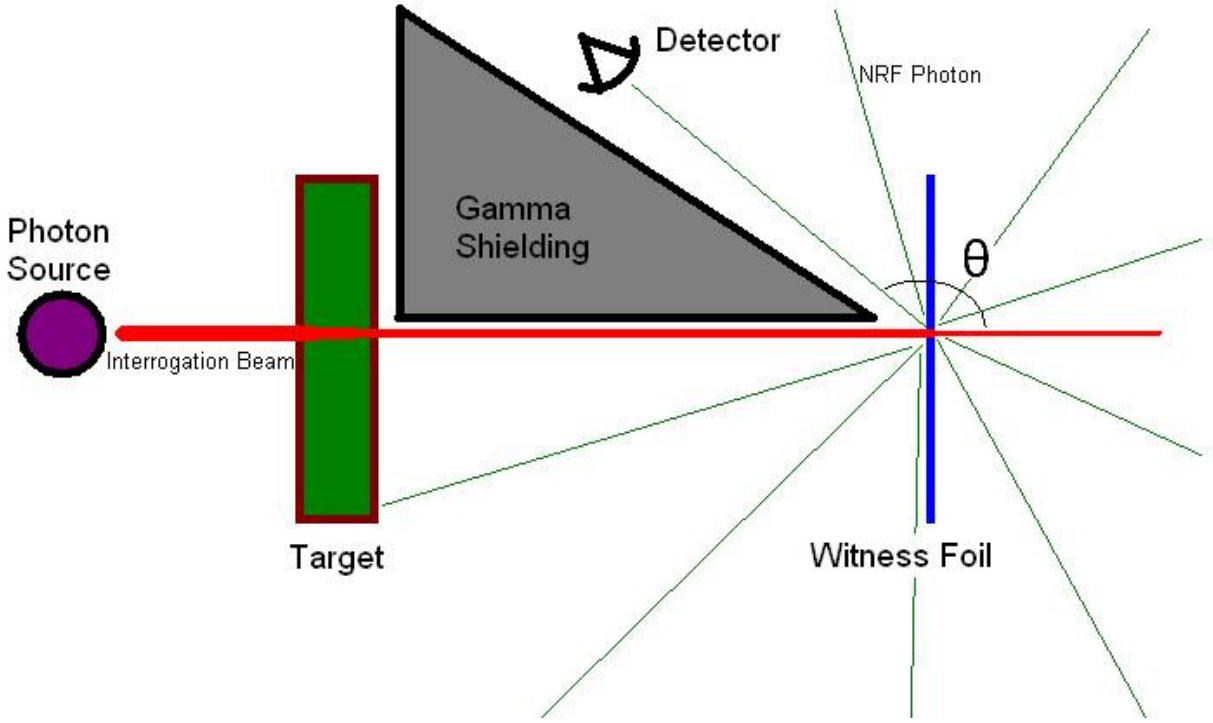
Instead of measuring the NRF spectrum directly, we can also use a tomography-like approach to measure the effects the target has on the transmitted interrogation beam. This allows us to more accurately find the total amount of NRF absorption that occurred in the target. Since NRF radiation is emitted isotropically, the large majority of NRF photons will travel in a different direction than the original interrogating beam. A direct measurement of the transmitted beam shows how much of it was absorbed or scattered, thereby providing information on the nuclides present in the target. However, the interrogating beam itself is generally too intense for a direct measurement. Also, one is often concerned only with particular features of the output spectrum, i.e. at energies near the resonances of a specific nuclide.

A witness foil may be used to address the issues involved with direct measurement of the beam exiting the target. These foils are composed of a single type of nuclide (at least as pure as manufacturing allows), and therefore respond in a very well-defined manner to an interrogating beam. NRF in the foil is the dominant effect over an energy range comparable to the energy resolution in HPGe detectors (~3 keV). Therefore, one can expect significant differences in the emissions from the witness foil, depending on how the initial target affects the spectrum of the interrogation beam.

For example, consider two different targets and a test using a ^{235}U witness foil. One of the targets contains almost no ^{235}U, while the other is enriched to, say, 20%. A bremsstrahlung source creates a continuous photon spectrum with a large flux at 1.733 MeV (one of the excitation energies of ^{235}U). Unless ^{235}U is present in the target, very few of the 1.733 MeV photons in the beam interact within the target. Therefore, when this beam interacts with the low-enrichment target, it is mostly unattenuated at this key energy. When the beam then hits the witness foil, large NRF effects are observed. But in the high-enrichment target, a large fraction of the 1.733 MeV intensity is absorbed within the target. Thus, fewer of these photons remain to induce NRF within the foil. The emitted spectrum from the foil depends strongly on how much ^{235}U is present in the target.

THE PROBLEM

While it is clear that the NRF spectrum from a witness foil depends on the composition of the initial target, we need to know the relationship precisely to effectively scan nuclear material. Obtaining reliable analytical calculations is the first step towards this goal. Hand calculations for all but the simplest cases are untenable, due to the large number of nuclides involved and the complexity of cross-sections as functions of energy. MCNP seems the natural solution to get past the large amount of computation involved. However, we have recently uncovered flaws hardwired into MCNP's treatment of scattering physics. (See Brian Quiter's work on Rayleigh scattering for high-Z nuclei for a detailed description of the problem.) These flaws made MCNP unsuitable for our simulations.



THE SOLUTION

To work around these issues, we have written a set of Matlab tools designed to simulate a simple witness foil geometry. The code relies on user input for cross-section data; this allows one to include or remove various interactions at will by modifying the cross-sections. One must also specify the compositions of the target and the witness foil, among other physical parameters. A detailed description of the input can be found in the documentation section of this paper.

The toolset currently includes 6 Matlab functions:

- witnessCalc
  This is the main control interface. It reads cross-section data from an input file, calls methods to calculate attenuation coefficients, computes output spectra, and writes the results to a text file.
- fileReader
  Reads numerical data from a delimited text file into a matrix in Matlab. Note that the input file must be rectangular in that each line contains the same number of fields. The function skips any line containing non-numerical text.
- parseTXTline
  A helper method for fileReader, which reads a delimited text string and returns an array of values
- atomToMassPercent
  witnessCalc supports composition inputs in either relative atomic abundance or in mass percent. Since it is often easier to compute atomic abundances, this function converts the composition into mass percent form. This form is then directly used for calculating photon attenuation within both the target and foil.
- attenuator
  This function takes composition and cross-section data to compute attenuation coefficients as functions of energy.
- writeDataToTxt
  Matlab's data output is remarkably clunky and ill-suited for text files. This function allows one to more easily print data arrays directly to a specified text file.

COMPUTATIONAL METHODS

To compute the intensity of photons emitted by the witness foil, we must find the attenuation of the interrogation beam through each material. Suppose that a nuclide n_i has a cross-section function $\sigma_i(E)$ and a number density of N_i atoms per unit volume. Then the attenuation coefficient as a function of energy is

$$\mu_i(E) = \sigma_i(E)\, N_i$$

The total attenuation coefficient for the material is $\mu(E) = \Sigma\, \mu_i(E)$ summed over all nuclides present. Then, given an incident photon intensity of $I_0(E)$, the attenuated intensity of the output beam is

$$I(E) = I_0(E)\, e^{-\mu(E)\, x}$$

Where x is the length of the path the beam follows through the material. For normal incidence, x is simply the thickness of the material.

Note that the number density of nuclide n_i can be calculated from its atomic mass and the density of the solid material it is part of. Specifically, if ρ is the density of the material, A_i is the atomic mass, and M_i is the mass percent of the nuclide within the material,

$$N_i = (\rho\, N_A\, M_i\, /100)/\, A_i$$

where $N_A = 6.022 \times 10^{23}$ is Avogadro's number. The atomic abundance M_i' is related to the mass percent M_i by the formula

$$M_i = A_i\, M_i' \,/\, M$$

Where $M = \Sigma\, A_i\, M_i'$ summed over all nuclides present.

For our purposes, we assume the following geometry. The interrogating beam is incident normal to the target, which has a thickness D1 and attenuation coefficients μ1(E). Therefore, the spectrum of photons leaving the target (and hence which are incident on the witness foil) is defined by

$$I_{\text{to foil}}(E) = I_0(E)\, e^{-\mu 1(E)\, D1}$$

This spectrum is further attenuated in the foil to $I_{out}(E) = I_{\text{to foil}}(E)\, e^{-\mu 2(E)\, D2}$, given a foil thickness D2 and attenuation coefficients μ2(E). The amount of NRF produced in the foil during this attenuation is thus $I_{NRF}(E) = I_{\text{to foil}}(E) - I_{out}(E)$.

However, this NRF spectrum is itself attenuated as it exits the foil towards the detector. Since the NRF emissions occur at different depths within the foil, we must find an "average" attenuation distance. Let θ be the angle from the original beam axis in the foil to the detector. Assume that the distance to this detector is large enough relative to the thickness of the foil that θ does not change throughout the foil. By this, assume that the attenuation depth as seen by the NRF photons is constant over the whole solid angle subtended by the detector.

Define the average NRF intensity within the foil to be $I_{avg}(E) = (\, I_{\text{to foil}}(E) + I_{out}(E)\,)/2$. The effective attenuation coefficient due to the witness foil is

$$\mu_e(E) = \mu(E)\, (1 + |\cos\theta|^{-1})$$

Let the average attenuation depth then be defined

$$D(E) = -\, \mu_e(E)^{-1}\, \ln\, (I_{avg}(E))$$

Finally, the NRF intensity that emerges from the foil at an angle of θ is:

$$I_{final}(E) = I_{NRF}(E)\, e^{-\mu 2(E)\, D(E)}$$

Divide by 4π to get the intensity per steradian.

DOCUMENTATION

This section documents the use of witnessCalc, including input/output files and command lines. As mentioned in the introduction, this toolset includes 6 Matlab functions, all of which are required to run the chain properly. Below is a complete description of the command-line use for each function.

- [postTarg_spec,final_spec] =
  witnessCalc(inputTable,atomicOrMass,targ_comp,targ_thick,targ_dense,foil_comp,foil_thick,
  foil_dense,detectorAngle,output_filename)

  All parameters except output_filename are required. If output_filename is not given, no output file is written.
  - inputTable is a string that lists the name of main cross-section data file to be used. This file must be located in the current working directory in Matlab
  - atomicOrMass is a switch with two options: 'atomic' or 'mass'. This specifies if atomic abundance or mass percent was used to define compositions. Note that this option must be the same for both the foil and the target.
  - targ_comp is a three-column array detailing the composition of the target. It has the following format:
    - Column 1: 5-digit ZAID
    - Column 2: Relative composition (in atomic % or mass %)
    - Column 3: Column index in inputTable for cross-section data for the nuclide
  - targ_thick simply gives the path length (in cm) of the beam through the target
  - targ_dense is the mass density of the target material, in g/cm^3
  - foil_comp has the same form as targ_comp, but specifies the foil composition
  - foil_thick is the thickness of the witness foil in cm
  - foil_dense is the mass density of the foil material, in g/cm^3
  - detectorAngle is the angle between the interrogating beam direction and the line from the witness foil to the detector
  - output_filename is a string specifying the name of a tab-delimited text file to write output data to

  Output values
  - postTarg_spec is the relative intensity ($I_{to\ foil}/I_0$) of the photon beam that leaves the back of the target and is thus incident on the target.
  - final_spec is the relative intensity of the NRF lines that exit the witness foil. It is given in (I_{final}/I_0) / steradian.

  Both output values are given as column vectors. The output written to output_filename includes a 5-column array of the form:
  - Column 1: Energy (eV)
  - Column 2: Source Intensity (in #/cm^2/s/eV)
  - Column 3: Bin width for the current row, in eV
  - Column 4: postTarg_spec (normalized to source intensity)
  - Column 5: final_spec (normalized to source intensity, per steradian)
  - Column 6: Absolute NRF spectra leaving the foil (per steradian)

  In addition, the compositions and physical parameters for the foil and target, and a copy of inputTable are included.

- dataArray = fileReader(filename,spacingChar)
  The filename parameter is required; spacingChar is optional and defaults to ',' if not specified.
  - filename is a string giving the name of a text file to read into Matlab's memory.
  - spacingChar represents the character used to delimit the text file. '\t' (tabs) and ',' (commas) are the most common.
  - dataArray is the Matlab matrix of numerical data as retrieved from the file in question.

  Note that inputTable must be a *.csv file within this toolchain. fileReader is called using the default comma delimiter.

- [dataFields] = parseTXTline(line,spacingChar)
  The filename parameter is required; spacingChar is optional and defaults to ',' if not specified.
  - line is a delimited text string to separate into a vector of fields
  - spacingChar represents the character used to delimit the text file.
  - dataFields is a Matlab vector of string data as retrieved from the text line in question.

- massPerc = atomToMassPercent(atomPerc)
  - atomPerc is a composition array, as used by witnessCalc. It contains 3 columns as follows:
    - Column 1: 5-digit ZAID
    - Column 2: Relative composition (in atomic %)
    - Column 3: Column index in inputTable for cross-section data for the nuclide
  - massPerc is the same composition array as atomPerc, except that column 2 now lists the composition in mass percents

- coefficients = attenuator(composition,density,masterArray)
  - composition is a three column array with the standard form used for compositions in this toolset. The columns are:
    - Column 1: 5-digit ZAID
    - Column 2: Relative composition (in mass %)
    - Column 3: Column index in inputTable for cross-section data for the nuclide
  - density is the material mass density of the material through which a beam is attenuated
  - masterArray is the numeric array of cross-sections as functions of energy, as read from inputTable in witnessCalc
  - coefficients is a vector of attenuation coefficients as a function of energy

- writeDataToFile(filename,spacingChar,varargin)
  - filename is a string that gives the name of the file to write output data to.
  - spacingChar specifies the delimiting character to write the output with
  - varargin can be any number of cells, strings, or arrays to be written to an output file.

  Note that newlines are automatically inserted between each dataset given in varargin. To insert special characters or extra delimiters, one must write them directly into the arrays.

The main cross-section data is taken from a comma-separated value (*.csv) file whose name is specified by the inputTable argument in witnessCalc. The structure of this file is as follows:

- Column 1: Energy in keV
- Column 2: Interrogation beam intensity, in #/cm^2/s/eV
- Column 3: Bin width for the current row, in eV
- Columns 4+: Cross-sections (in barns) for the nuclides used in the calculation

EXAMPLE USE

To illustrate the use of witnessCalc, here is an example computation. The inputs below are written in the form one would use in Matlab.

```
targ_comp =
    [8016,55.1,7;40090,17.2,6;92238,27.7,4];

foil_comp = [92235,100,5];

[postTarg_spec,final_spec] =
    witnessCalc('masterArray.csv','atomic',targ_comp,21.8,4,foil_comp,0.5,19.1,135,
    'witnessCalcData.txt');
```

Here, the target is composed of 55.1% ^{16}O, 17.2% ^{90}Zr, and 27.7% ^{238}U by atomic abundance. Similarly, the foil is 100% pure ^{235}U. The target is 21.8 cm thick and has density 4 g/cm^3. The foil is 5 mm thick and has a density of g/cm^3. The detector is at a backwards angle of 135˚. Finally, all output is written to a file called witnessCalcData.txt in the current working directory.

The compositions above state that the cross section data for ^{16}O can be found in column 7 of 'masterArray.csv'. Similarly, ^{90}Zr cross sections are in column 6, while those for ^{238}U and ^{235}U are in columns 4 and 5, respectively.

In addition, the masterArray.csv file might be:

```
Energy (eV),Intensity (#/eV),binWidth (eV),U-238 XS (b),U-235 XS (b),Zr-90 XS (b),O-16 XS (b)
1731500,1e9,1498,2,2,2,2
1732998,9e8,1,2,2.5,2,2
1732999,9e8,1,2,5,2,2
1733000,9e8,1,2,10,2,2
1733001,9e8,1,2,5,2,2
1733002,9e8,1,2,2.5,2,2
1734500,8e8,1498,2,2,2,2
```

In more readable form, this translates to:

Energy (eV)	Intensity (#/eV)	binWidth (eV)	U-238 XS (b)	U-235 XS (b)	Zr-90 XS (b)	O-16 XS (b)
1731500	1.00E+09	1498	2	2	2	2
1732998	9.00E+08	1	2	2.5	2	2
1732999	9.00E+08	1	2	5	2	2
1733000	9.00E+08	1	2	10	2	2
1733001	9.00E+08	1	2	5	2	2
1733002	9.00E+08	1	2	2.5	2	2
1734500	8.00E+08	1498	2	2	2	2

This table shows that ^{235}U has an NRF peak of 10 barns centered at 1.733 MeV, where the initial beam intensity gives 9.00E+08 photons/cm^2/s. Note that in the first and last bins, which are much wider than the others, many more photons are created. This allows one to weight the spectrum appropriately to deal with extremely fine energy resolution and bulk bins at the same time.

SOURCE CODE

The source code for witnessCalc, fileReader, parseTXTline, atomToMassPercent, attenuator, and writeDataToTxt is given below.

```
function writeDataToFile(filename,spacingChar,varargin)
% Function writes a tab-delimited file from data specified in varargin

% -- "filename" is the name of the file to be written
% -- "spacingChar" determines how the data fields in each object passed
%    through varargin should be spaced. For example, if spacingChar = '\t',
%    then the output file will be tab delimited.
% -- varargin contains any number of headers and/or data matrices to write
%    to a file. This function decides what each parameter is and prints it
%    appropriately.

fileID = fopen(filename,'wt');

% First, we determine how many objects there are to print
for i = [1:length(varargin)]
   data = varargin{i};
   % Matlab is very clunky for dealing with file output and strings
   % We need to convert each entry in data into a string to write
   % separately.
   [rows,columns] = size(varargin{i});
   for R =[1:rows]
      temp = cell(1,columns);
      for C = [1:columns]
         temp = data(R,C); % temp is the ith row of the dataArray matrix
         if isa(temp,'numeric') == 1
            temp = num2str(temp);
         elseif isa(temp,'cell') == 1
               temp = temp{1};
         else
            error(['Undefined data type. Cannot write as text']);
         end
         % Special characters will be interpreted literally. Let's
         % interpret them if they come up:
         if strcmp(temp,'\n')
            fprintf(fileID,'\n');
         elseif strcmp(temp,'\t')
            fprintf(fileID,'\t')
         % Print the data and a tab-delimiter, unless it's the last column
         elseif C == columns % Print a newline character instead of a tab here
            fprintf(fileID,'%s\n',temp);
         else
            fprintf(fileID,'%s\t',temp);
         end
      end
   end
end
fclose(fileID);
```

```
% Reads numerical data from a *.csv file into memory
% This function automatically removes any lines that contain non-number
% strings. Make sure the *.csv is rectangular and only contains one dataset!

function dataArray = fileReader(filename,spacingChar)
fileID = fopen(filename);
if fileID < 0
  error(['Could not open ',filename,' for input']);
end

% If spacingChar is not specified, this reads *.csv files by default
if nargin < 2
   spacingChar = ',';
end
% First we need to figure out the dimensions of our master array.
% "rows" represents the number of rows, while "cols" is the number of
% columns
status = 1;
rows = 0;
cols = 0;
while status > 0
   line = fgetl(fileID);
   % We only want to see how many fields are in the line
   junk = parseTXTline(line,spacingChar);
   tempCols = length(junk);
   % If this line has a different number of fields than the line before it,
   % we have a problem!
   if (tempCols ~= cols && cols ~= 0 && tempCols ~= 0)
      error(['*.csv data is not rectangular']);
   end
   if line == -1
      status = 0;
      break
   end
   rows = rows + 1;
   cols = tempCols;
end

% Need to reinstantiate the *.csv file for access
fileID = fopen(filename);

% strArray is a cell whose entries represent the columns in the *.csv file
% It is output as a cell of strings
strArray = cell(rows,cols);

% This loop populates strArray and then checks to see which rows have
% non-number strings in them
badRows = []; % Contains the row indices for the non-number strings
for i = [1:rows]
   line = fgetl(fileID);
   NaNFlag = 0;
   [data,junk] = strtok(line,';');
   % Now we parse the data by comma-delimiting
   fields = parseTXTline(data,spacingChar);
   for j = [1:cols]
      % Is the data point not a valid number?
```

```
      if size(str2num(fields{j})) == [0 0]
         NaNFlag = 1; % True if one or more entries is NaN
      end
      strArray{i,j} = fields{j};
   end
   % If we found a non-number, we'll skip the row for the final array
   if NaNFlag == 1
      badRows = [badRows i];
   end
end

% We have now read the *.csv file into memory. Now we want to take out
% header rows or any other rows containing text
[junk,rowsToDel] = size(badRows);
dataArray = zeros(rows-rowsToDel,cols);
index = 1;
for i = [1:rows]
   % Checking if badRows contains the current row index
   % Only writes good rows to dataArray
   if size(find(badRows == i)) ~= [1 1]
      for j = [1:cols]
         dataArray(index,j) = str2num(strArray{i,j});
      end
      index = index + 1;
   end
end

fclose all;
```

```
function [dataFields] = parseTXTline(line,spacingChar)
% Uses a while loop to extract data fields from a comma-delimited string

% If spacingChar is not specified, this reads *.csv files by default
if nargin < 2
   spacingChar = ',';
end

% Want to figure out how many data entries there are, first
% This loop finds the first field from the string, then loops using the line
% minus that field
counter = 0;
tempLine = line;
while length(tempLine) > 1
   [field,remainder] = strtok(tempLine,spacingChar);
   tempLine = remainder;
   counter = counter + 1;
end

dataFields = cell(1,counter);
% Must reinitialize the line to read into dataArray
tempLine = line;
for i = 1:counter
   [field,remainder] = strtok(tempLine,spacingChar);
   tempLine = remainder;
   dataFields{i} = field;
end
```

```
function massPerc = atomToMassPercent(atomPerc)
% Converting composition arrays from atomic percent to mass percent

% This function converts the material composition given by the
% atomPerc matrix, specified in atomic percentages, into the
% masses matrix. The output lists the composition in terms of mass
% percents.

% The input matrix should have two columns: the first gives ZAIDs for the
% component nuclides, while the second gives the atomic percentage within
% the material in question.

[nuclides,columns] = size(atomPerc);

% massPerc has an identical structure to atomPerc. Only column 2 is
% different
massPerc = atomPerc;
masses = zeros(nuclides,1);

totalAtomPercent = 0; % A quick check to make sure the composition was normalized correctly
for i = [1:nuclides]
   % What is the number density of the nuclide?
   ZAID = atomPerc(i,1);

   Z = double(uint16(ZAID/1000));
   A = ZAID - Z*1000;
   atomPercent = atomPerc(i,2);

   totalAtomPercent = totalAtomPercent + atomPercent;

   % Now to find the weighted mass proportions
   masses(i) = atomPercent*A;
end

% The total mass is the sum of all entries in masses; this lets us
% compute mass percents
totalMass = sum(masses);
massPerc(:,2) = 100*masses/totalMass;

if totalAtomPercent ~= 100
   sprintf('Warning: An atomic composition is not normalized to 100 percent!\nIt sums to %f percent.',totalAtomPercent)
end
```

```
function coefficients = attenuator(composition,density,masterArray)
% Used in tandem with witnessCalc to compute attenuation coefficients

% The input requires the composition, density, and thickness of the
% material through which a beam is attenuated. In addition, a masterArray
% contains all the cross-section data for the nuclides specified in the
% composition matrix.

%
% -- composition is a three-column matrix specifying the composition of the
%    target. The first column should list ZAIDs while the second lists the
%    relative abundance of nuclides (in mass %). The third column points to
%    the column in inputTable that contains cross-section data for the
%    nuclide.
% -- density is the density of the target in g/cc

% masterArray has the following structure:
% Column 1: Bin energy in keV
% Column 2: Intensity of the source as a function of energy within each bin
%          (in units of #/s/cm^2/eV for normalization)
% Column 3: Widths of the energy bins in column 2 (in eV)
% Columns 4+: Cross-sections of nuclides in the target or foil as a
%           function of energy (in barns)

[N_bins,columns] = size(masterArray); % Looks up how many energy bins there are

% Need to find the size of targ_comp to know how many nuclides are present
[nuclides,columns] = size(composition);

% The number of interactions in the target as a function of E, per unit time
coefficients = zeros(N_bins,1);

totalMassPercent = 0; % A quick check to make sure the composition was input correctly
for i = [1:nuclides]
   % What is the number density of the nuclide?
   ZAID = composition(i,1);
   Z = double(uint16(ZAID/1000));
   A = ZAID - Z*1000;

   massPercent = composition(i,2);

   N_density = (6.022*10^23)*(density*massPercent/100)/A;

   % Where is the cross-section data in masterArray?
   xs_col = composition(i,3);
   nuclide_xs = 10^-24*masterArray(:,xs_col);

   % Each individual component of the target contributes to the effective
   % cross section, and hence the attenuation coefficient:
   coefficients = coefficients + N_density*nuclide_xs;
end
```

```
function writeDataToFile(filename,spacingChar,varargin)
% Function writes a tab-delimited file from data specified in varargin

% -- "filename" is the name of the file to be written
% -- "spacingChar" determines how the data fields in each object passed
%    through varargin should be spaced. For example, if spacingChar = '\t',
%    then the output file will be tab delimited.
% -- varargin contains any number of headers and/or data matrices to write
%    to a file. This function decides what each parameter is and prints it
%    appropriately.

fileID = fopen(filename,'wt');

% First, we determine how many objects there are to print
for i = [1:length(varargin)]
   data = varargin{i};
   % Matlab is very clunky for dealing with file output and strings
   % We need to convert each entry in data into a string to write
   % separately.
   [rows,columns] = size(varargin{i});
   for R =[1:rows]
      temp = cell(1,columns);
      for C = [1:columns]
         temp = data(R,C); % temp is the ith row of the dataArray matrix
         if isa(temp,'numeric') == 1
            temp = num2str(temp);
         elseif isa(temp,'cell') == 1
               temp = temp{1};
         else
            error(['Undefined data type. Cannot write as text']);
         end
         % Special characters will be interpreted literally. Let's
         % interpret them if they come up:
         if strcmp(temp,'\n')
            fprintf(fileID,'\n');
         elseif strcmp(temp,'\t')
            fprintf(fileID,'\t')
         % Print the data and a tab-delimiter, unless it's the last column
         elseif C == columns % Print a newline character instead of a tab here
            fprintf(fileID,'%s\n',temp);
         else
            fprintf(fileID,'%s\t',temp);
         end
      end
   end
end
fclose(fileID);
```